Have a look at my previous post, Passing Data Transfer Objects with GET in Spring Boot for information about how the DTOs are being passed to the Rest Controller. Also have a look at Spring's starter guide if your starting from scratch. The setup that is not described in this post is covered there.

The only Maven dependency required in this post is

```
1  <dependencies>
2     <dependency>
3        <groupId>org.springframework.boot</groupId>
4        <artifactId>spring-boot-starter-web</artifactId>
5     </dependency>
6  </dependencies>
```

view raw spring-boot-dto-tutorial.xml hosted with ❤ by **GitHub**

The DTO

```
1  public class PersonDTO {
2
3     private String firstName;
4     private String secondName;
5     // Formats output date when this DTO is passed through JSON
6     @JsonFormat(pattern = "dd/MM/yyyy")
7     // Allows dd/MM/yyyy date to be passed into GET request in JSON
8     @DateTimeFormat(pattern = "dd/MM/yyyy")
9     private Date dateOfBirth;
10
11    private String profession;
12    private BigDecimal salary;
13
14    public PersonDTO(
15          String firstName, String secondName, Date dateOfBirth, String profession, BigDecimal salary) {
16       this.firstName = firstName;
17       this.secondName = secondName;
18       this.dateOfBirth = dateOfBirth;
```

```
19      this.profession = profession;
20      this.salary = salary;
21    }
22
23    public PersonDTO() {}
24
25    public String getFirstName() {
26      return firstName;
27    }
28
29    public void setFirstName(String firstName) {
30      this.firstName = firstName;
31    }
32
33    public String getSecondName() {
34      return secondName;
35    }
36
37    public void setSecondName(String secondName) {
38      this.secondName = secondName;
39    }
40
41    public Date getDateOfBirth() {
42      return dateOfBirth;
43    }
44
45    public void setDateOfBirth(Date dateOfBirth) {
46      this.dateOfBirth = dateOfBirth;
47    }
48
49    public String getProfession() {
```

```
50      return profession;
51    }
52
53    public void setProfession(String profession) {
54      this.profession = profession;
55    }
56
57    public BigDecimal getSalary() {
58      return salary;
59    }
60
61    public void setSalary(BigDecimal salary) {
62      this.salary = salary;
63    }
64  }
```

view raw PersonDTO.java hosted with ❤ by **GitHub**

A few things to notice

```
@JsonFormat(pattern = "dd/MM/yyyy")
```

Formats the date when the DTO is output to JSON. If this is not used the JSON will display a number that represents the time instead of a easy to read string.

```
@DateTimeFormat(pattern = "dd/MM/yyyy")
```

This works the other way around as it allows the date to be input in *dd/MM/yyyy* format, which if your trying to pass a date directly into JSON it will be hard to know the number version of the date you want.

Also remember to include your default constructor, getters and setters otherwise serializing and deserializing of the DTO will not work.

The controller

```
1  @RestController
2  public class PersonRestController {
3
4    private static final SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd/MM/yyyy");
5
6    @RequestMapping("/getPersonDTO")
7    public PersonDTO getPersonDTO(@RequestParam(value = "personDTO") String jsonPersonDTO)
8        throws IOException {
9     return getPersonDTOFromJson(jsonPersonDTO);
10   }
11
12   private PersonDTO getPersonDTOFromJson(final String jsonPersonDTO) throws IOException {
13    return new ObjectMapper()
14       .setDateFormat(simpleDateFormat)
15       .readValue(jsonPersonDTO, PersonDTO.class);
16   }
17
18   @RequestMapping("/getPersonDTOList")
19   public List<PersonDTO> getPersonDTOList(
20       @RequestParam(value = "personDTO") String jsonPersonDTO,
21       @RequestParam(value = "personDTO2") String jsonPersonDTO2)
22       throws IOException {
23     final PersonDTO personDTO = getPersonDTOFromJson(jsonPersonDTO);
24     final PersonDTO personDTO2 = getPersonDTOFromJson(jsonPersonDTO2);
25     return Arrays.asList(personDTO, personDTO2);
26   }
27
28   @RequestMapping("/getPeopleDTO")
29   public PeopleDTO getPeopleDTO(
30       @RequestParam(value = "personDTO") String jsonPersonDTO,
```

```
31      @RequestParam(value = "personDTO2") String jsonPersonDTO2)
32      throws IOException {
33    final PersonDTO personDTO = getPersonDTOFromJson(jsonPersonDTO);
34    final PersonDTO personDTO2 = getPersonDTOFromJson(jsonPersonDTO2);
35    return new PeopleDTO(Arrays.asList(personDTO, personDTO2));
36   }
37  }
```

view raw PersonRestControllerForReturningDTO.java hosted with ❤ by **GitHub**

Each of these methods are returning DTOs but in slightly different ways. The nice thing about reaching this point is that all the configuration for returning a *PersonDTO* has already been done and there is nothing fancy that needs to be done now.

Lets look at each method individually.

```
1  @RequestMapping("/getPersonDTO")
2  public PersonDTO getPersonDTO(@RequestParam(value = "personDTO") String jsonPersonDTO)
3     throws IOException {
4   return getPersonDTOFromJson(jsonPersonDTO);
5  }
```

view raw GetPersonDTO.java hosted with ❤ by **GitHub**

Nothing interesting happens in this code. The input *personDTO* from the request is parsed into a *PersonDTO* object and returned. As I mentioned above all the setup for returning the *PersonDTO* has already been done due to the code added to it's class. This includes the getters and setters and the *@JsonFormat* which allows it to be returned with its values and have the date field formatted nicely.

So if we pass a request to to the controller (I used Postman to do this) we can see what happens.

```
localhost:8080/getPersonDTO?personDTO={"firstName":"First name","secondName":"Second
name","profession":"Professional time waster","salary":0,"dateOfBirth":"01/012/2020"}
```

Which outputs the following JSON that represents the *PersonDTO*.

```
{
 "firstName": "First name",
 "secondName": "Second name",
 "dateOfBirth": "01/12/2020",
 "profession": "Professional time waster",
 "salary": 0
}
```

Notice that the *dateOfBirth* data is a string formatted to *dd/MM/yyyy* due to the *@JsonFormat* that was added to the field in the *PersonDTO*.

Does anything interesting happen if we try to return a *List<PersonDTO>* ? Lets have a look.

```
1  @RequestMapping("/getPersonDTOList")
2  public List<PersonDTO> getPersonDTOList(
3     @RequestParam(value = "personDTO") String jsonPersonDTO,
4     @RequestParam(value = "personDTO2") String jsonPersonDTO2)
5     throws IOException {
6    final PersonDTO personDTO = getPersonDTOFromJson(jsonPersonDTO);
7    final PersonDTO personDTO2 = getPersonDTOFromJson(jsonPersonDTO2);
8    return Arrays.asList(personDTO, personDTO2);
9  }
```

view raw GetPersonDTOList.java hosted with ♥ by **GitHub**
Called with the request.

```
localhost:8080/getPersonDTOList?personDTO={"firstName":"First name","secondName":"Second
name","profession":"Professional time waster","salary":0,"dateOfBirth":"01/12/2020"}&personDTO2=
{"firstName":"Random first name","secondName":"Random second name","profession":"Professional
sleeper","salary":123,"dateOfBirth":"11/12/2100"}
```

Which leads leads to the JSON output.

```
[
 {
  "firstName": "First name",
  "secondName": "Second name",
  "dateOfBirth": "01/12/2020",
  "profession": "Professional time waster",
  "salary": 0
 },
 {
  "firstName": "Random first name",
  "secondName": "Random second name",
  "dateOfBirth": "11/12/2100",
  "profession": "Professional sleeper",
  "salary": 123
 }
]
```


As you can see from the output each *PersonDTO* is contained within the square brackets that represent the list in JSON. So did anything interesting happen in the code or the return data? Nope, still nice a simple.

The last example is slightly different from first looks but it is pretty much the same as returning the *List<PersonDTO>*.

```
1  public class PeopleDTO {
2
3    private List<PersonDTO> people;
4
5    public PeopleDTO() {}
6
7    public PeopleDTO(List<PersonDTO> people) {
8      this.people = people;
9    }
10
11   public List<PersonDTO> getPeople() {
12     return people;
13   }
14
15   public void setPeople() {
16     this.people = people;
17   }
18 }
```

view raw PeopleDTO.java hosted with ❤ by **GitHub**

Now that you have seen the *PeopleDTO* code your understand why is so similar to the previous example as it is just an object that contains a *List<PersonDTO>*. You might prefer to return this DTO rather than a *List<PersonDTO>* but I wont make that decision for you.

Lets look at the controller code.

```
1  @RequestMapping("/getPeopleDTO")
2  public PeopleDTO getPeopleDTO(
3      @RequestParam(value = "personDTO") String jsonPersonDTO,
4      @RequestParam(value = "personDTO2") String jsonPersonDTO2)
5      throws IOException {
6    final PersonDTO personDTO = getPersonDTOFromJson(jsonPersonDTO);
7    final PersonDTO personDTO2 = getPersonDTOFromJson(jsonPersonDTO2);
8    return new PeopleDTO(Arrays.asList(personDTO, personDTO2));
9  }
```

view raw GetPeopleDTO.java hosted with ❤ by **GitHub**

Send it the request.

```
localhost:8080/getPeopleDTO?personDTO={"firstName":"First name","secondName":"Second
name","profession":"Professional time waster","salary":0,"dateOfBirth":"01/12/2020"}&personDTO2=
{"firstName":"Random first name","secondName":"Random second name","profession":"Professional
sleeper","salary":123,"dateOfBirth":"11/12/2100"}
```

And retrieve the JSON output.

```
{
 "people": [
  {
   "firstName": "First name",
   "secondName": "Second name",
   "dateOfBirth": "01/12/2020",
   "profession": "Professional time waster",
   "salary": 0
  },
  {
   "firstName": "Random first name",
   "secondName": "Random second name",
   "dateOfBirth": "11/12/2100",
   "profession": "Professional sleeper",
   "salary": 123
  }
 ]
}
```

Yet again the code required to set this up is pretty simple. Other than creating the *PeopleDTO* to store the *List<PersonDTO>* nothing else in the code has changed. The JSON output is slightly different from the previous example as the list is now tied to the *people* property.

So what did you learn from reading this post? Not much actually as returning a data transfer object from a Rest Controller is actually pretty straight forward. Simply set up your DTO correctly with a default constructor, getters and setters and maybe add some annotations if your feeling more sophisticated, after that there's not really anything left to do.

The code used in this post can be found on my GitHub.